MPLS VPN Functional Specification

Chapter Eight

VPN Manager

By: Richard Crump and Janet Doong

May 12, 2000

8.1 VPN Manager

In order to limit the impact of supporting multiple routing tables on existing protocol sub-systems the VPN Manager is introduced. The VPN Manager will manage the interaction between client protocol sub-systems and multiple routing tables. Interactions include adding, updating, and deleting routes. This will save developers from adding the same multiple routing table interface logic to all the clients. Clients will only need minor modifications to their existing routing table interface logic to interface with the VPN Manager.

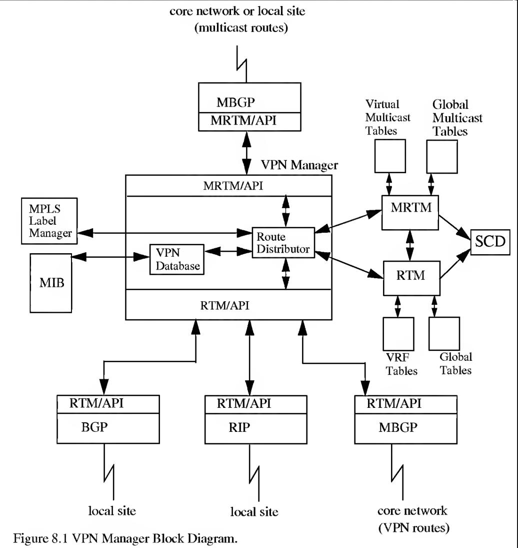

Figure 8.1 VPN Manager Block Diagram.

Figure 8.1 above illustrates how the VPN Manager interfaces with its protocol sub-system clients and with other system resources. The VPN Manager supports two application program interfaces (APIs), one for clients using the MRTM/API and one for clients using the RTM/API. It also interfaces to system services/resources including the MPLS Label Manager, the system

MIBs, VRFs, and the global and multicast routing tables. Internally to the VPN Manager is a VPN Database and Route Distributor logic.

VPN Database is a database built from reading the system's configuration MIBs. It is indexed by VPN ID and will relate which VRFs are associated with each VPN.

RTM/API is used by clients, such as RIP and BGP, that currently interact with the global routing table via RTM. The existing API is expanded to include an indication of which routing table (global routing table or one of the VRFs) the routing table operation is for.

MRTM/API is used by clients, such as MBGP and DVMRP, that currently interact with the multicast routing table via MRTM. The existing API is expanded to include an indication of which routing table (global multicast table or one of the virtual multicast tables) the routing table operation is for.

Route Distributor is the heart of the VPN Manager. It will have the intelligence to know if a routing table operation needs to be duplicated on other tables or not. If operation needs to be duplicated it will be able to know which other routing tables to apply it on.

8.1.1 Remote Route Distribution

This section discusses the simple case of learning a local route by a PE router with only one CE interface. See the next section for how routes learnt by a PE router with multiple CE interfaces are distributed to all the local VRFs.

Figure 8.2 VPN and Global Route Distribution Flow Diagram is provided to illustrate the following discussion on how a local route learnt by the PE 1 Router is distributed to the PE 2 Router.

On the PE 1 Router, when BGP, for example, submits a route to the VPN Manager via the RTM/API, the VPN Manager will determine if this route was learnt from a VPN network or a classical IP network.[1] If it was learnt from a classical IP network, it will be submitted to the Global Routing Table, RTM. Otherwise, the VPN Manager will obtain a MPLS Label from the MPLS Label Manager for this route. Then the VPN Manager will submit the route to the VRF associated with the VPN network the route was learnt from. In addition, the VPN Manager will operate in such a way as to let MBGP learn the route.

On the PE 2 Router, MBGP will submit its VPN routes to the VPN Manager via the RTM/API. The VPN Manager will look up in its VPN Database to find all VRFs having intersecting VPN membership with the route's Target VPN set. The VPN Manager will then submit the routes to each of the those VRFs.

1. This is done by referring to the value of the VRF_ID included in the IP_RT_ENTRY that is submitted via the API call.

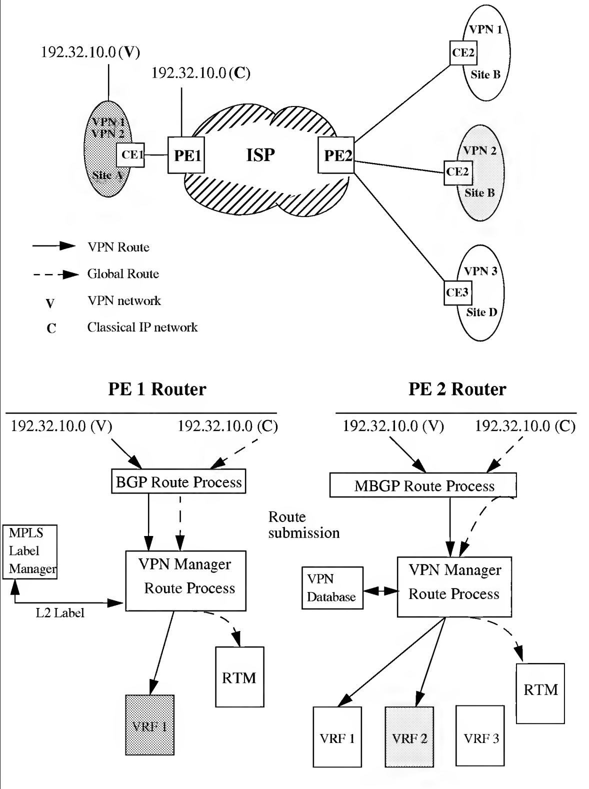

Figure 8.2 VPN and Global Route Distribution Flow Diagram.

8.1.2 Local Route Distribution

This section discusses how local routes learnt by a PE router with multiple CE interfaces are distributed to all the local VRFs. See the previous section for how local routes learnt on a PE router with only one CE interface are distributed to a remote PE router.

Figure 8.3 VPN Route Local Distribution Flow Diagram is provided to illustrate the following discussion on how a local route learnt by the PE Router is distributed between multiple local VRFs.

When BGP, for example, submits a route to the VPN Manager via the RTM/API, the VPN Manager will determine if this route was learnt from a VPN network or a classical IP network[2]. If it was learnt from a classical IP network, it will be submitted to the Global Routing Table, RTM. Otherwise, a MPLS Label will be obtained from the MPLS Label Manager for this route. The VPN Manager will find the VPN membership associated with this network, and look up in its VPN Database to find which VRFs have intersecting memberships with this route. The VPN Manager will then submit the route to each of the VRFs. In addition, the VPN Manager will operate in such a way as to let MBGP learn the route only from the VRF associated with the interface the route was learnt from.

8.1.3 MIB Considerations

When a restart attribute of the VRF MIB record is changed all routes in the associate VRF routing table will be deleted. The routing table itself will be deleted and then recreated using the values from the updated VRF MIB record.

8.1.4 MPLS Label Manager API

The VPN Manager will conform to the API provide by the MPLS sub-system. For a description of this API, see the MPLS Label Manager API specification.

8.1.5 Switch Control Driver (SCD) API

The VPN Manager will conform to the Switch Control Interface (SCI) provide by the SCD sub-system. For a description of this API, see the SCD API specification.

2. This is done by referring to the value of the VRF_ID included in the IP_RT_ENTRY that is submitted via the API call.

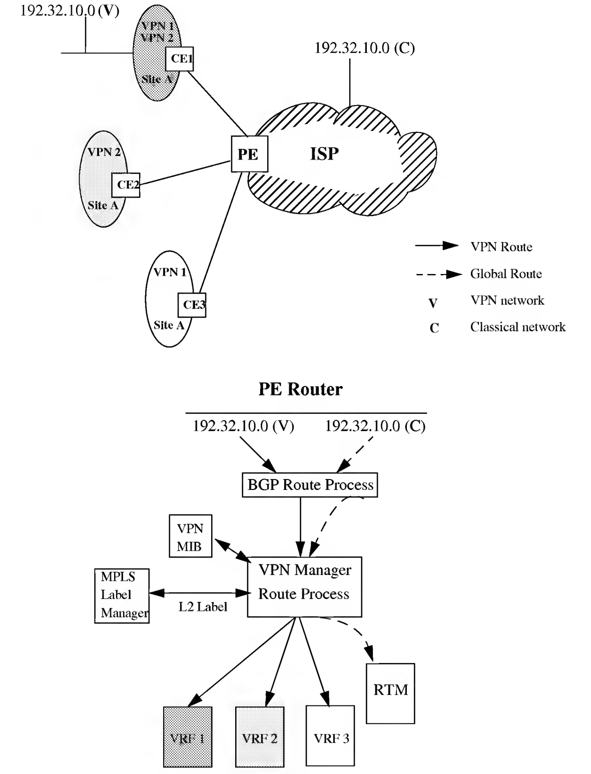

Figure 8.3 VPN Route Local Distribution Flow Diagram.

8.1.6 Other Considerations

The following points are outside the scope of the VPN Manager itself, but are needed for the proper operations of the VPN Manager.

First, the routine that currently creates the global routing table will need to read the VPN MIB records and create multiple VRF routing tables. Also this routine may also create the VPN Database used by the VPN Manager.

Second, the RTM Gate environment data structure needs in addition to its single routing pool pointer an array data structure supporting multiple VRF tables.

```
typedef struct vrf_data
    {
      R_TBL            *routing_table;      /* back pointer to VRF routing table      */
      T_wfVpnRecord    *vpn_cfg;            /* need to match new MIB record.          */
    } VRF_DATA;
```

Third, higher layer protocols, such as RIP, that support multiple interfaces, and thus, potentially multiple VPNs and VRFs, must register multiple times with the VPN Manager, once for each VRF.

Forth, IP's mappings of higher layer protocols will need to be modified to clean-up protocol specific routes in the VRFs, just like what is currently done to clean-up RTM.

8.2 RTM/Application Program Interface

VPN Manager clients, such as BGP and RIP, will use the RTM/API to interact with RTM routing table periodically when they receive or send route update packets. If any client needs to acquire the changed route information when routes are updated by routing protocols, then the client has to register as a sequence client.

The RTM/API includes the following routines:

```
ip_sequence_register()
ip_sequence_de_register()
ip_rtm_consider_net()
ip_rtm_consider_net_vpn()
ip_rtm_consider_host()
ip_net_find_exact()
ip_net_find_exact_vpn()
ip_route_find()
ip_get_next_rnp_queue()
ip_rtm_refresh_rt_seq()
ip_valid_net_route()
```

All current IP related MACROs used to operate on the routing data structures will be support by VPN Manager. Also, VPN Manager will provide three new MACROs as follows:

```
IP_GET_VRF_DATA( rtm_env, vrf_index )
IP_GET_VRF_TABLE( rtm_env, vrf_index )
IP_GET_VRF_CFG( rtm_env, vrf_index )
```

The invocation of the following MACROs that use "rtm_env->route_pool" will need to be modified to use a pointer to the proper routing table (VRF or Global). The routing table pointer can be found by invoking one of the above new MACROs.

```
r_balance()
r_delete()
r_find_exact()
r_next_range()
r_range32()
r_range32_gap()
FOR_EACH_R_RECORD()
FOR_EACH_R_RECORD_IN_RANGE()
R_FIRST_RECORD()
R_KEY_CMP()
```

8.2.1 ip_sequence_register()

SYNOPSIS

```
RNP_ENTRY *ip_sequence_register(    u_int32 *seq_num_ptr,
                                    u_int32 hold_down_time,
                                    u_int32 book_flag,
                                    u_int32 bookmark_id,
                                    u_int32 vrf_index)
```

WHERE

seq_num_ptr - a pointer to the client sequence number storage allocation.

hold_down_time - registration block is inserted in the list of clients in descending order of hold down time.

book_flag - If clients want to traverse the routing tables then they need to set book_flag to TRUE otherwise set it to FALSE.

bookmark_id - Client also has to tell the VPN Manager its unique bookmark_id, see the following bookmark_id defined values.

```
#define IP_RTM_BGP_BOOKMARK_ID          0x10000000
#define IP_RTM_BGP_CONN_BOOKMARK_ID     0x20000000
#define IP_RTM_OSPF_BOOKMARK_ID         0x40000000
#define IP_RTM_DVMRP_BOOKMARK_ID        0x01000000
#define IP_RTM_RSVP_BOOKMARK_ID         0x02000000
#define IP_RTM_MON_BOOKMARK_ID          0x04000000
#define IP_RTM_PIM_BOOKMARK_ID          0x08000000
```

vrf_index - index into IP's VRF_DATA array. The array elements contains a pointer to the specific routing table that the client wants to monitor. Clients can pass -1 to indicate they want to monitor all VRF tables together as one large virtual routing table. Note, index value of 0 (zero) is reserved for the Global Routing Table.

DESCRIPTION

The VPN Manager will return a bookmark (pointer to a dummy RNP_ENTRY) to its clients indicating a successful registration. Clients can visit changed routes through use of their bookmarks by calling RTM/API utility routines.

ALSO SEE

ip_sequence_de_register(), ip_get_next_rnp_queue()

8.2.2 ip_sequence_de_register()

SYNOPSIS

void ip_sequence_de_register(RNP_ENTRY *bookmark)

WHERE

bookmark - pointer to the bookmark returned when the client registered as a sequence client.

DESCRIPTION

When clients have no more interest in the routing tables, clients need to call ip_sequence_de_register(). The client has to pass in their bookmark to the de-register routine so the VPN Manager is able to remove client's information and client's installed routes from routing tables.

The VPN Manager also maps to each individual client gate. If a client dies, the VPN Manager will clean up all the routes installed by this client and will clean up the client's bookmark too.

When routes are removed from the routing tables the associated L2 labels (the MPLS/BGP VPN inner label) are released via call to the MPLS Label Manager.

ALSO SEE

ip_sequence_register()

8.2.3 ip_rtm_consider_net()

SYNOPSIS

```
RNP_ENTRY *ip_rtm_consider_net (   RT_ENTRY *nrt_entry,
                                   RTM_ENV *rtm_env,
                                   u_int32 add_type,
                                   RNP_ENTRY *rnp_entry)
```

WHERE

nrt_entry -pointer to a new routing entry to consider.

rtm_env - pointer to RTM Gate's environment.

add_type - Tells how to consider this addition.

```
#define ADD_LOCAL      1            /* Just keep it locally     */
#define ADD_GLOBAL     2            /* Send it to everybody     */
#define SLOT_DOWN      0x80000000   /* when a slot dies         */
#define IGNORE_ADD     3            /* Don't add this entry     */
```

rnp_entry - pointer to a routing network pool entry (may be null).

DESCRIPTION

After clients have registered, then they are able to submit their routes via the RTM/ API by calling the ip_rtm_consider_net() routine. Based on the value of the vrf_index element of the IP_RT_ENTRY being submitted, the VPN Manager is able to add/ delete/update routes in the appropriate routing tables.

If a locally learned route is being submitted to a VRF, the VPN Manager will obtain a MPLS Label from the MPLS Label Manager for this route.

If the route is being deleted from a VRF, then the associated L2 label (the MPLS/BGP VPN inner label) is released via call to the MPLS Label Manager.

See next page for RT_ENTRY and IP_RT_ENTRY data structures.
Also see, ../include/ip_rtm.h file for additional details on the data structures.

ALSO SEE

ip_net_find(), ip_net_find_exact(), ip_get_next_rnp_queue()

```
typedef struct rt_entry
    {
      IP_ADDRESS    address;            /* The address of this network              */
      IP_ADDRESS    mask;               /* The network mask of the network          */
      IP_RT_ENTRY   ip_rt_entry;
    } RT_ENTRY;
```

```
typedef struct ip_rt_entry
    {
      ip_rt_entry       *next_best;         /* ptr to next best route                         */
      u_int32           record_type;        /* Must be RT_ENTRY_TYPE                          */
      RT_WEIGHT         weight;             /* route weight                                   */
      u_int32           author;             /* The author of this entry                       */
      u_int32           rt_flags;           /* Flags associated with the entry                */
      u_int32           lifetime;           /* This is filled in when the route is added      */
      u_int8            slot;               /* Slot that authored this                        */
      u_int8            protocol;           /* The protocol it came from                      */
      u_int8            fake_gh_type;       /* For fake hosts, what is the gh type            */
      u_int8            rip_hold_done;      /* it's been held down                            */
      IP_ADDRESS        nexthop;            /* The nexthop address for this range             */
      IP_ADDRESS        nexthop_interface;  /* nexthop interface for this range               */
      u_int32           age;                /* Age since last update                          */
      u_int32           rip_hold_time;      /* timer for rip hold down                        */
    union                                   /* interpretation is specific to the protocol     */
    {
      IP_ADDRESS        ospf_id;            /* if OSPF, ID of router that authored this route */
      u_int32           egp_autonomous;     /* if EGP, autonomous system                      */
      u_int32           bgp_next_hop_as;    /* if BGP, next hop AS number                     */
      u_int32           static_rt_id;       /* if STATIC, the instance ID, for ECMP           */
    } specific;
      u_int32           ospf_tag;           /* OSPF tag                                       */
      TBLOCK            birth_time;         /* the time the route was submitted to RTM        */
      FWD_ENTRY         *fwd_entry;         /* point to fwd_entry_queue for ECMP              */
      RTM_SUB_REG *register_block;          /* point to RTM register block for non-children */
                                            /* of IP protocol only                            */
      u_int32           host_hdl;           /* the host hdl for the nexthop                   */
      u_int32           vrf_index;          /* route from local site.                         */
      u_int32           l2_label;           /* MPLS/VPN inner label                           */
    } IP_RT_ENTRY;
```

8.2.4 ip_rtm_consider_net_vpn()

SYNOPSIS

```
RNP_ENTRY *ip_rtm_consider_net_vpn (  RT_ENTRY *nrt_entry,
                                      RTM_ENV *rtm_env,
                                      u_int32 add_type,
                                      RNP_ENTRY *rnp_entry,
                                      VPN_TARGET *vpn_target)
```

WHERE

nrt_entry -pointer to a new routing entry to consider.

rtm_env - pointer to RTM Gate's environment.

add_type - Tells how to consider this addition.

```
#define ADD_LOCAL      1            /* Just keep it locally     */
#define ADD_GLOBAL     2            /* Send it to everybody     */
#define SLOT_DOWN      0x80000000   /* when a slot dies         */
#define IGNORE_ADD     3            /* Don't add this entry     */
```

rnp_entry - pointer to a routing network pool entry (may be null).

vpn_target - pointer to a VPN target list.

DESCRIPTION

After clients have registered, then they are able to submit their VPN-IPv4 routes via the RTM/API by calling the ip_rtm_consider_net_vpn() routine. Based on the VPN target list, the VPN Manager is able to add/delete/update routes in the appropriate routing tables.

When VPN-IPv4 routes are submitted, the routes are duplicated, if necessary, and stored in VRFs with VPN membership matching/overlapping with the VPN target list.

```
typedef struct vpn_target
   {
     u_int32        vpn_target_count;     /* length of VPN Target list.              */
     u_int32        vpn_target[1];        /* list of VPNs this route is for          */
   } VPN_TARGET;
```

ALSO SEE

ip_net_find_exact(), ip_get_next_rnp_queue()

8.2.5 ip_rtm_consider_host()

SYNOPSIS

```
RNP_ENTRY *ip_rtm_consider_host ( HT_ENTRY_UPDATE *ht_entry,
                                  RTM_ENV *rtm_env,
                                  u_int32 add_type,
                                  RHP_ENTRY *rhp_entry)
```

WHERE

ht_entry -pointer to a new host entry to consider.

rtm_env - pointer to RTM Gate's environment.

add_type - Tells how to consider this addition.

```
#define ADD_LOCAL     1            /* Just keep it locally    */
#define ADD_GLOBAL    2            /* Send it to everybody    */
#define SLOT_DOWN     0x80000000   /* when a slot dies        */
#define IGNORE_ADD    3            /* Don't add this entry    */
```

rhp_entry - pointer to a routing host pool entry (may be null).

DESCRIPTION

After clients have registered, then they are able to submit hosts via the RTM/API by calling the ip_rtm_consider_host() routine. Based on the value of the vrf_index element of the HT_ENTRY_UPDATE being submitted, the VPN Manager is able to add/delete/update hosts in the appropriate routing tables.

If a locally learned host is being submitted to a VRF, the VPN Manager will obtain a MPLS Label from the MPLS Label Manager for this host.

If the host is being deleted from a VRF, then the associated L2 label (the MPLS/BGP VPN inner label) is released via call to the MPLS Label Manager.

See next page for HT_ENTRY_UPDATE data structure.
Also see, ../include/ip_rtm.h file for additional details on the data structures.

ALSO SEE

```
typedef struct ht_entry_update
    {
      u_int32          record_type;          /* Must be RT_ENTRY_TYPE                   */
      RT_WEIGHT        weight;               /* route weight                            */
      u_int32          author;               /* The author of this entry                */
      u_int32          rt_flags;             /* Flags associated with the entry         */
      u_int32          lifetime;             /*  This is filled in when the route is added    */
      u_int8           slot;                 /* Slot that authored this                 */
      u_int8           protocol;             /* The protocol it came from               */
      u_int16          pad1;                 /* for cr34754                             */
      IP_ADDRESS       address;              /* The address of this network/host        */
      GH               dest_gh;              /* The destination gate id for this range  */
      u_int8           dest_gh_type;         /* Which type of gh is it? Static/Dynamic  */
      u_int8           other_gh_type;        /* Which type of gh is it? Static/Dynamic  */
      u_int8           fake_gh_type;         /* Which type to use for FAKE ness         */
      u_int8           mux;                  /* the mux scheme for this host            */
      IP_ADDRESS       nexthop;              /* The nexthop address for this range      */
      IP_ADDRESS       nexthop_interface;    /* nexthop interface addr for this range   */
      nwif_env         *nexthop_nwif;        /* The nexthop network interface           */
      MADR48           mac_addr;             /* The mac address of this host.           */
      u_int16          pad;                  /* pads out MAC address for to 64 bits.    */
      u_int32          x121_pad[2];          /* room for x121 address.                  */
      u_int32          ip_mtu;               /* max size of an IP packet to this host.  */
      u_int16          nexthop_cct;          /* Nexthop CCt: Host Only and Unnumbered use */
      u_int8           ttl;
      u_int8           delete_state;
      RTM_SUB_REG *register_block;           /* point to RTM register block for non-children */
                                             /* of IP protocol only.                    */
      u_int32          host_hdl;
      u_int32          encaps_hdl;
      u_int32          vrf_index;            /* route from local site.                  */
      u_int32          l2_label;             /* MPLS/VPN inner label                    */
    } HT_ENTRY_UPDATE;
```

8.2.6 ip_net_find_exact()

SYNOPSIS

```
RNP_ENTRY *ip_net_find_exact (   u_int32 vrf_index,
                                 IP_ADDRESS address,
                                 IP_ADDRESS mask)
```

WHERE

vrf_index - index into IP's VRF_DATA array. The array elements contains a pointer to the specific routing table that the client wants to search. Note, index value of 0 (zero) is reserved for the Global Routing Table.

address - The address of the network to find.

mask - The network mask of the network address.

DESCRIPTION

ip_net_find_exact() routine will return a RNP_ENTRY pointer if an exact match is found in the VPN Manager routing table for the requested address and mask, otherwise a NULL pointer is returned.

The first calling parameter was the rtm_env pointer in the original RTM/API.

If MBGP needs support for this type of function while processing VPN-IPv4 routes then we will need to modify parameter list to include a VPN Target or provide a separate API routine that is passed VPN Target instead of a routing table pointer.

ALSO SEE

ip_net_find(), ip_get_next_rnp_queue(), ip_rtm_consider_net()

```
typedef struct rnp_entry
   {
     R_HEADER        r_header;
     FWD_ENTRY       embedded_fwd_entry;  /* default block for uni-path                 */
     u_int32         flags;               /* Flags associated with the entry            */
     FWD_ENTRY       *fwd_entry;          /* Address of Distilled forwarding version    */
     IP_RT_ENTRY     *best_route;         /* best route pointer                         */
     queue_data_t    queue;               /* double linked list of RNP_ENTRY            */
     aging_queue_entry *ageing_ptr;       /* Ptr to corresponding block on ageing queue */
     u_int32         *v;                  /* pointer to corresponding OSPF LSDB element */
     u_int32         lsid;                /* LSID of the corresponding LSDB             */
     u_int32         sequence;            /* unique RTM sequence number                 */
     u_int16         was_prot;            /* protocol id of the best route              */
     u_int16         book_flags;          /* flags: see../include/ip_rtm.h file         */
     u_int16         mp_proxy_cnt;        /* Next one to be proxied                     */
     u_int16         mp_count;            /* Number of equal cost multipath entries     */
     u_int8          black_hole_count;    /* BGP connections that used this black hole  */
     u_int8          last;                /* last one used when round robin in effect   */
     u_int8          reserve;             /* Declared to maintain 4 byte boundary       */
     u_int32         host_hdl;            /* the host handle this address is associated with */
   } RNP_ENTRY;
```

8.2.7 ip_net_find_exact_vpn()

SYNOPSIS

```
RNP_ENTRY *ip_net_find_exact (   VPN_TARGET *vpn_target,
                                 IP_ADDRESS address,
                                 IP_ADDRESS mask)
```

WHERE

vpn_target - pointer to a VPN target list that the client wants to search.

address - The address of the network to find.

mask - The network mask of the network address.

DESCRIPTION

ip_net_find_exact_vpn() routine will return a RNP_ENTRY pointer if an exact match is found in a VRF that has a VPN membership matching/overlapping with the VPN target list, for the requested address and mask, otherwise a NULL pointer is returned.

The first calling parameter was the rtm_env pointer in the original RTM/API.

ALSO SEE

ip_get_next_rnp_queue(), ip_rtm_consider_net_vpn()

8.2.8 ip_route_find()

SYNOPSIS

RNP_ENTRY *ip_route_find (**u_int32 vrf_index,**
IP_ADDRESS address)

WHERE

vrf_index - index into IP's VRF_DATA array. The array elements contains a pointer to the specific routing table that the client wants to search. Note, index value of 0 (zero) is reserved for the Global Routing Table.

address - The address of the network to find.

DESCRIPTION

ip_route_find() routine will return the most specific RNP_ENTRY for a given network address. If no entry is found then a NULL pointer is returned.

The first calling parameter was the rtm_env pointer in the orginal RTM/API.

ALSO SEE

ip_net_find_exact(), ip_valid_net_route(), ip_get_next_rnp_queue(), ip_rtm_consider_net()

8.2.9 ip_valid_net_route()

SYNOPSIS

```
RNP_ENTRY *ip_valid_net_route ( u_int32 vrf_index,
                                RTM_ENV *rtm_env,
                                IP_ADDRESS address)
```

WHERE

vrf_index - index into IP's VRF_DATA array. The array elements contains a pointer to the specific routing table that the client wants to search. Note, index value of 0 (zero) is reserved for the Global Routing Table.

rtm_env - pointer to RTM Gate's environment.

address - The destination address to find.

DESCRIPTION

ip_valid_net_route() routine will search for the most specific net route that should be used for forwarding to a given destination address. If it finds an unreachable route that is used for hole punching in an aggregate route (e.g. a host that could not be resolved by ARP or a destination that is explicitly advertised as unreachable), that route is returned instead of a less specific reachable route. If the search fails a NULL pointer is returned.

ALSO SEE

ip_route_find(), ip_rtm_consider_host()

8.2.10 ip_get_next_rnp_queue()

SYNOPSIS

```
RNP_ENTRY *ip_get_next_rnp_queue(     RNP_ENTRY *bookmark,
                                      RTM_ENV *rtm_env,
                                      u_int32 skip_flags)
```

WHERE

bookmark - pointer to the bookmark returned when the client registered as a sequence client.

rtm_env - pointer to RTM Gate's environment.

skip_flags - set to TRUE if client does no want to skip over BGP Main Gate bookmark (IP_RTM_BGP_BOOKMARK_ID).

DESCRIPTION

ip_get_next_rnp_queue() routine will return the next available real/non-client bookmark RNP_ENTRY from the VNP Manager's routing tables.

Only locally learned routes are available as changed routes associated with the client's bookmark. Routes that were locally distributed or learned from MBGP are not available as changed routes. This is to prevent a route from being advertised more than once as a side effect of being in multiple VRFs.

ALSO SEE

ip_net_find(), ip_net_find_exact(), ip_rtm_consider_net(), ip_rtm_refresh_rt_seq()

8.2.11 ip_rtm_refresh_rt_seq()

SYNOPSIS

```
RNP_ENTRY *ip_rtm_refresh_rt_seq(RTM_ENV *rtm_env,
                                  RNP_ENTRY *rnp_entry)
```

WHERE

rtm_env - pointer to RTM Gate's environment.

rnp_entry - pointer to a routing network pool entry.

DESCRIPTION

ip_rtm_refresh_rt_seq() routine will allow clients to request the VPN Manager to reprocess this route by moving it to the tail of VPN Manager's route sequence queue.

ALSO SEE

ip_get_next_rnp_queue()

8.3 MRTM/Application Program Interface

VPN Manager clients, such as MBGP, PIM and DVMRP, that use the MRTM/API they have to map to MRTM well-known UP gate. Clients are able to monitor MRTM state changes by the triggered mapping routines of MRTM UP gate. Clients can register to VPN Manager only when they detect MRTM state is MRTM_API_REG or MRTM_TABLE_READY.

There are two types of VPN Manager clients using the MRTM/API.

They are unicast route dependent multicast clients such as PIM and MSDP, or other clients such as MBGP and DVMRP. For the former type of the clients, a list of interested unicast route types such as OSPF, RIP, best route, etc needs to be provided to VPN Manager by the clients during registration.

If any client needs to acquire the changing route information of when routes are updated by routing protocols, then the client has to register using a bookmark_id and a valid sequence number pointer.

The MRTM/API includes the following routines:

mrtm_sequence_register()
mrtm_sequence_de_register()
mrtm_consider_net()
mrtm_route_find_exact()
mrtm_route_find()
mrtm_get_next_mrt_entry()
mrtm_get_next_best_mrt_entry()

The MRTM/API will not be active during the initial implementation of the VPN Manager since multicast protocol support is unavailable on the BFR.

8.3.1 mrtm_sequence_register()

SYNOPSIS

```
MRT_ENTRY *mrtm_sequence_register( u_int32 *seq_num_ptr,
                                   u_int32 book_flag,
                                   u_int32 bookmark_id,
                                   u_int32 ucast_flag,
                                   u_int32 *ucast_rt_type)
```

WHERE

seq_num_ptr – a pointer to the client sequence number storage allocated and initialized by the VPN Manager.

book_flag – If clients wanted to traverse the routing tables then they need to set book_flag to TURE otherwise set it to FALSE.

bookmark_id – Client also has to tell the VPN Manager its unique bookmark_id, see the following bookmark_id defined values.

```
#define MRTM_MROUTE_CLEANUP_BOOKMARK_ID                 0x00000000
#define MRTM_MROUTE_CHANGE_BOOKMARK_ID                  0x10000000
#define MRTM_MBGP_BOOKMARK_ID                           0x20000000
#define MRTM_MSDP_BOOKMARK_ID                           0x40000000
#define MRTM_PIM_BOOKMARK_ID                            0x80000000
#defineMRTM_DVMMRTM_CLIENT_NO_ROUTE_BOOKMARK_ID  0x02000000
#define RP_BOOKMARK_ID                                  0x01000000
```

ucast_flag – The client needs to set the ucast_flag parameter to TRUE, if it is a unicast route dependent type of clients.

ucast_rt_type – Clients have to pass in a route type array. The array is index by the route protocol type values defined in ip_rt_types.h. Each element set to TRUE indicates the clients interest in that type of route.

DESCRIPTION

The VPN Manager will return a bookmark to its clients indicating a successful registration. Clients can visit changed routes through use of their bookmarks by calling different MRTM/API utility routines.

8.3.2 mrtm_sequence_de_register()

SYNOPSIS

void mrtm_sequence_de_register(MRT_ENTRY *bookmark)

WHERE

bookmark - pointer to the bookmark returned when the client registered as a sequence client.

DESCRIPTION

When clients have no more interest in the routing tables, clients need to call mrtm_sequence_de_register(). The client has to pass in their bookmark_ptr to the de-register routine so the VPN Manager is able to remove client's information and the related routes from routing tables.

The VPN Manager also maps to each individual client gate. If a client died, the VPN Manager will clean up all the routes installed by this client and will clean up the client's bookmark too.

8.3.3 mrtm_consider_net()

SYNOPSIS

```
void mrtm_consider_net (MROUTE_INFO *mroute_info,
                        u_int32 injected_weight)
```

WHERE

mroute_info - is pointer to a data structure that carries information describing the route being submitted. See typedef below for details.

injected_weight - used as a new adjusted weight. If client doesn't need to change the preference or cost for the route, then this field will be set to the original route weight.

DESCRIPTION

After clients have registered, then they are able to submit their routes via the MRTM/API by calling the mrtm_consider_net() routine provided by the VPN Manager. Based on the clients' route submitted information, VPN Manager is able to add/delete/update its routing tables.

```
typedef struct mroute_info
  {
    IP_ADDRESS    net;
    IP_ADDRESS    mask;
    MRT_WEIGHT    weight;
    IP_ADDRESS    nexthop;              /* route source address                    */
    IP_ADDRESS    nexthop_interface;    /* route received interface address        */
    u_int32       protocol_specific;    /* 1.OSPF,ID of router that authored this route */
                                        /* 2.EGP, autonomous system                */
                                        /* 3.BGP, next hop AS number               */
                                        /* 4.STATIC, the instance ID, for ECMP     */
    u_int32       rt_flag;
    MROUTE_FWD_INFO fwd_entry;
    MROUTE_FWD_INFO embedded_fwd_entry; /* uni-path forwarding entry              */
    u_int8        slot;
    u_int8        protocol;             / * route type in ip_rt_types.h            */
    #define   UNICAST_BEST_ROUTE    0x0001
    #define   MRTM_DIRECT_ROUTE     0x0002
    #define   REMOVED_DIRECT_ROUTE  0x0004
    u_int16       route_flag;

  } MROUTE_INFO;
```

8.3.4 mrtm_route_find_exact()

SYNOPSIS

```
MRNP_ENTRY *mrtm_route_find_exact (  MRTM_ENV *mrtm_env,
                                     IP_ADDRESS address,
                                     IP_ADDRESS mask)
```

WHERE

mrtm_env - pointer to MRTM Gate's environment.

address - The address of the network to find.

mask - The network mask of the network address.

DESCRIPTION

mrtm_route_find_exact() routine will return a MRNP_ENTRY pointer if an exact match is found in the MRTM routing table for the requested address and mask, otherwise a NULL pointer is returned.

8.3.5 mrtm_route_find()

SYNOPSIS

```
MRNP_ENTRY *mrtm_route_find ( MRTM_ENV *mrtm_env,
                              IP_ADDRESS address)
```

WHERE

mrtm_env - pointer to MRTM Gate's environment.

address - The address of the network to find.

DESCRIPTION

mrtm_route_find() routine will return the most specific MRNP_ENTRY for a given network address. If no entry is found then a NULL pointer is returned.

8.3.6 mrtm_get_next_mrt_entry()

SYNOPSIS

MRT_ENTRY *mrtm_get_next_mrt_entry(MRT_ENTRY *bookmark)

WHERE

bookmark - pointer to the bookmark returned when the client registered as a sequence client.

DESCRIPTION

mrtm_get_next_mrt_entry() routine will return a next available real/non-client MRT_ENTRY.

8.3.7 mrtm_get_next_best_mrt_entry()

SYNOPSIS

MRT_ENTRY *mrtm_get_next_best_mrt_entry(MRT_ENTRY *bookmark)

WHERE

bookmark - pointer to the bookmark returned when the client registered as a sequence client.

DESCRIPTION

mrtm_get_next_best_mrt_entry() routine will return the next best route's MRT_ENTRY.